Web услуги. GET и POST заявки.

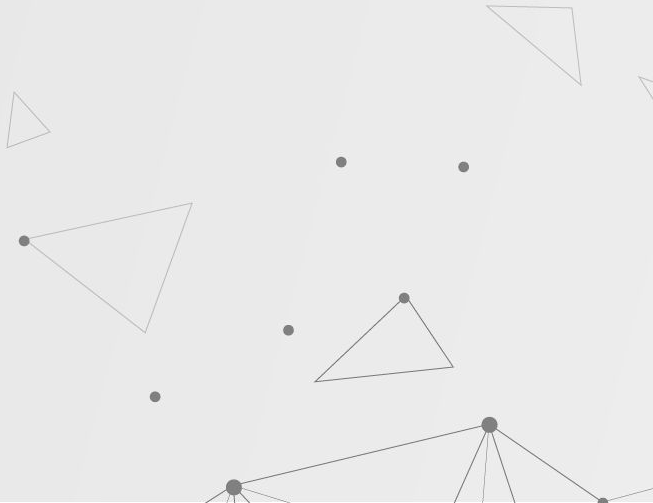
Иван Арабаджийски

Client-server архитектура.

HTTP Request.

HTTP Request Method.

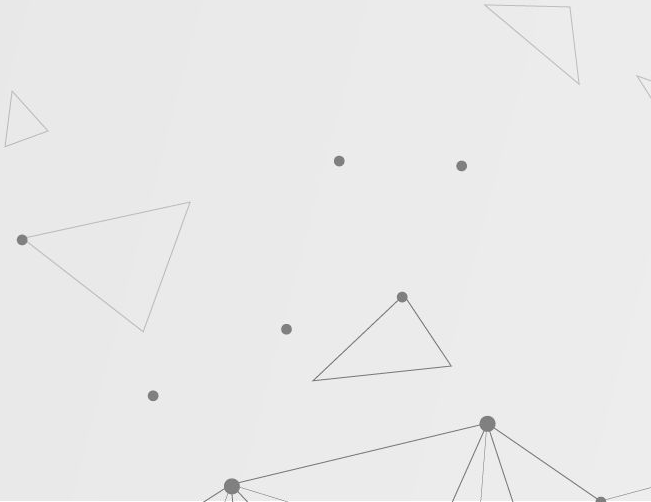
API



REST услуги.

Софтуерен архитектурен стил, създаден от Roy Fielding през 2000 година. Всеки API, който следва REST шаблона за дизайн се нарича RESTful API.

По-просто казано REST API е лесен начин два компютъра да комуникират помежду си чрез HTTP по същия начин, по който комуникират клиентите и сървърите.



Абревиатура?

- REpresentational
- State
- Transfer

Какво е RESTful API?

Множество от архитектурни ограничения.

Какво не е REST

Протокол или стандарт.

Какво прави едно API RESTful?

- Клиент-сървър архитектура съставена от клиенти, сървъри и ресурси, които се достъпват през HTTP.
- Stateless клиент-сървър комуникация. Това означава, че информация не се запазва между отделните get заявки и всяка заявка е отделна и несвързана с другите.

Всеки ресурс се определя по следния начин:

- Основен URI, например `http://api.example.com/`
- Стандартен HTTP метод - `GET, POST, PUT, and DELETE`
- Тип на ресурса

Добри практики

Използваме JSON формат за изпращане и получаване на данни

JavaScript има вече готови начини методи за работа с JSON обекти (fetch API), защото те са основно направени за нея. Естествено всички други програмни езици като Python и PHP също имат библиотеки и методи за за JSON данни.

За да сме сигурни, че клиентите интерпретират JSON данните правилно трябва да сложим `Content-Type: application/json`

в рекуест хедъра

От страната на сървъра доста фреймъурци вече правят това автоматично. Например `express.json()` `middleware`

Използваме съществителни вместо глаголи в имената на endpoint-ите

Това е така, защото всеки ресурс се определя от път и метод. Методите вече са с имена глаголи - `GET, POST, PUT, PATCH, and DELETE`

Това са основните CRUD (create, read, update, delete) операции.

Пример: един endpoint не трябва да изглежда така

`` `https://mysite.com/getPosts` or `https://mysite.com/createPost` ``

А по-скоро така:

`` `https://mysite.com/posts` ``

Накратно, трябва да оставим HTTP методите да описват какво прави endpoint-а. GET взима данни, POST създава данни, PUT променя, DELETE изтрива.

Имената на колекциите да са в множествено число

```
https://mysite.com/post/123 -> https://mysite.com/posts/123
```

Правилни статус кодове в обработка на грешки

STATUS CODE RANGE	MEANING
100-199	Informational Responses For example, 102 indicates the resource is being processed
300-399	Redirects For example, 301 means Moved permanently
400-499	Client-side errors 400 means bad request and 404 means resource not found
500-599	Server-side errors For example, 500 means an internal server error

Използване на вложени пътища за да покажем релация

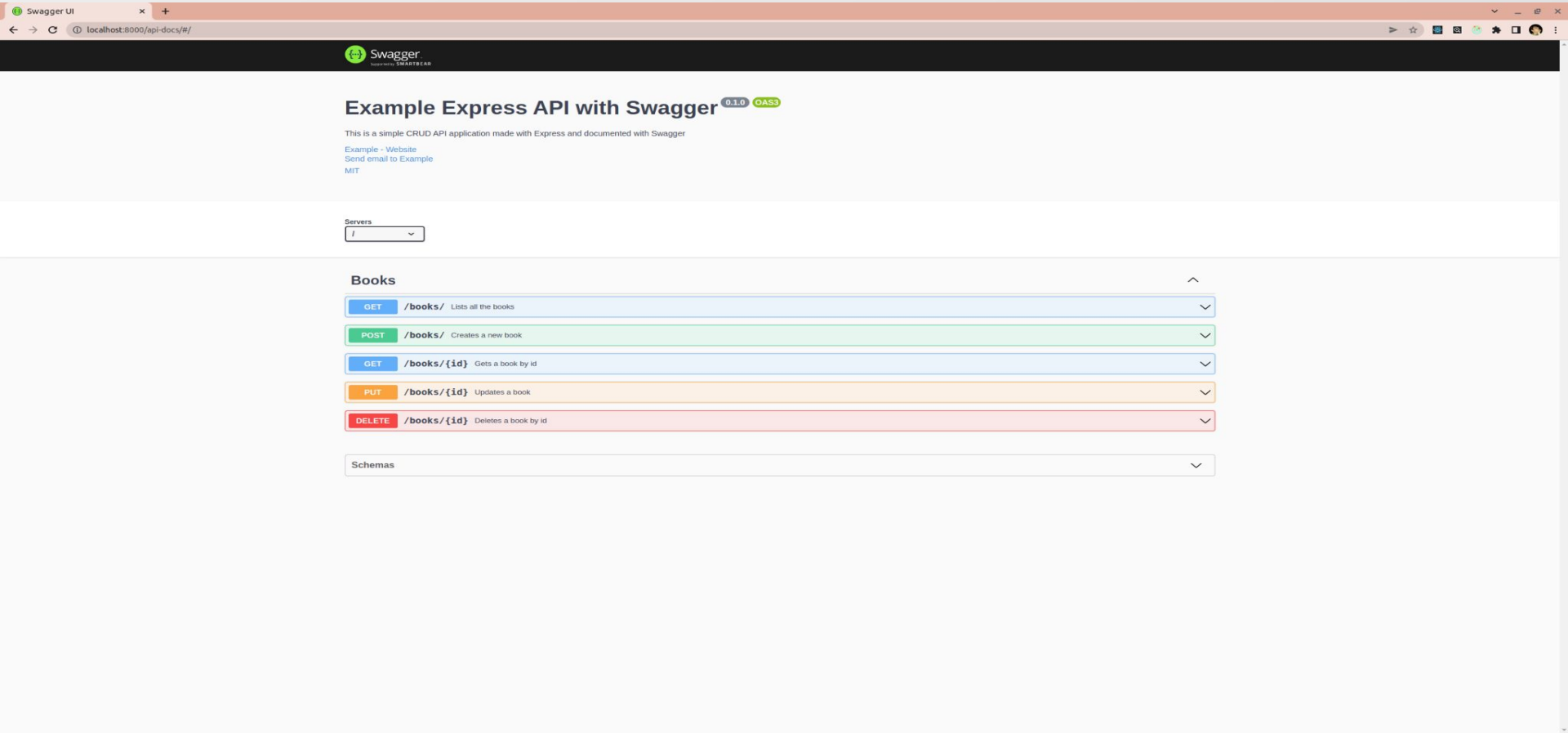
`https://mysite.com/posts/author` - Би било валидно влагане.

`https://mysite.com/posts/postId/comments` също

Използване на филтриране, сортиране и номериране на страници за да вземем необходимите данни

```
`https://mysite.com/posts?tags=javascript`
```

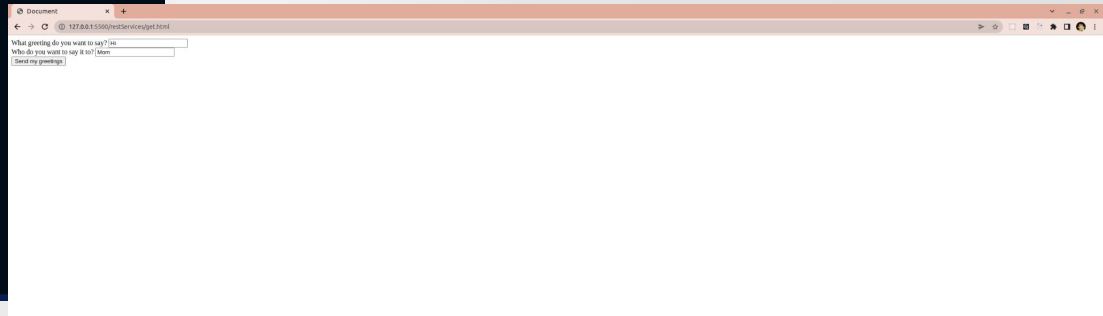
Добра документация



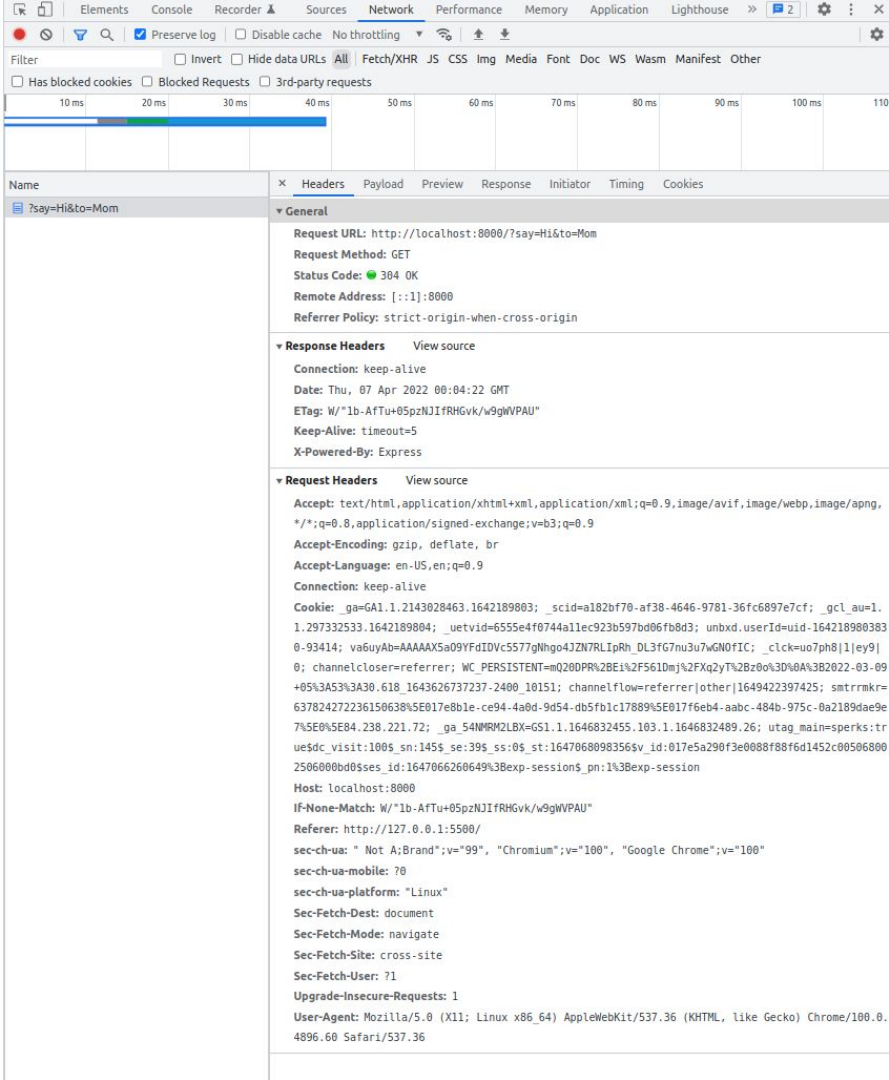
Изпращане на данни чрез HTML форми

GET

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <form action="http://localhost:8000/" method="GET">
      <div>
        <label for="say">What greeting do you want to say?</label>
        <input name="say" id="say" value="Hi" />
      </div>
      <div>
        <label for="to">Who do you want to say it to?</label>
        <input name="to" id="to" value="Mom" />
      </div>
      <div>
        <button>Send my greetings</button>
      </div>
    </form>
  </body>
</html>
```

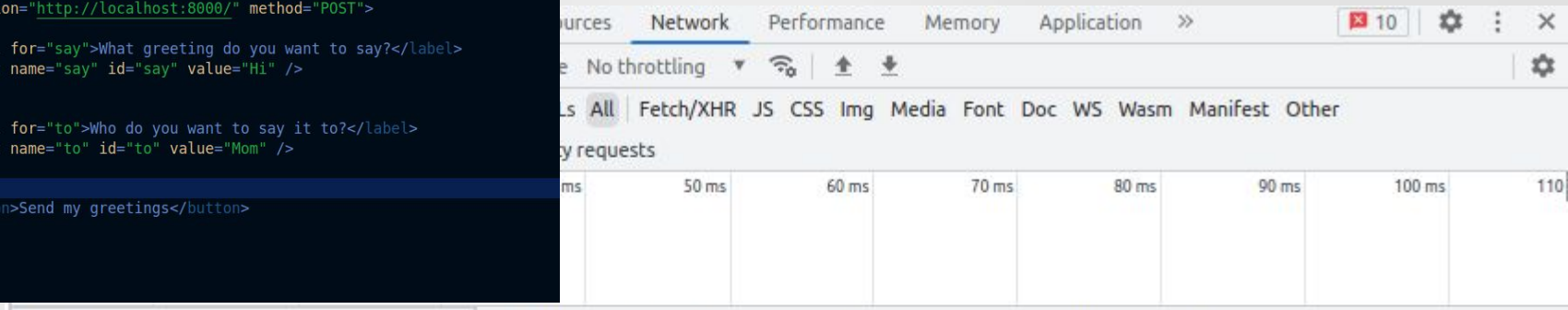


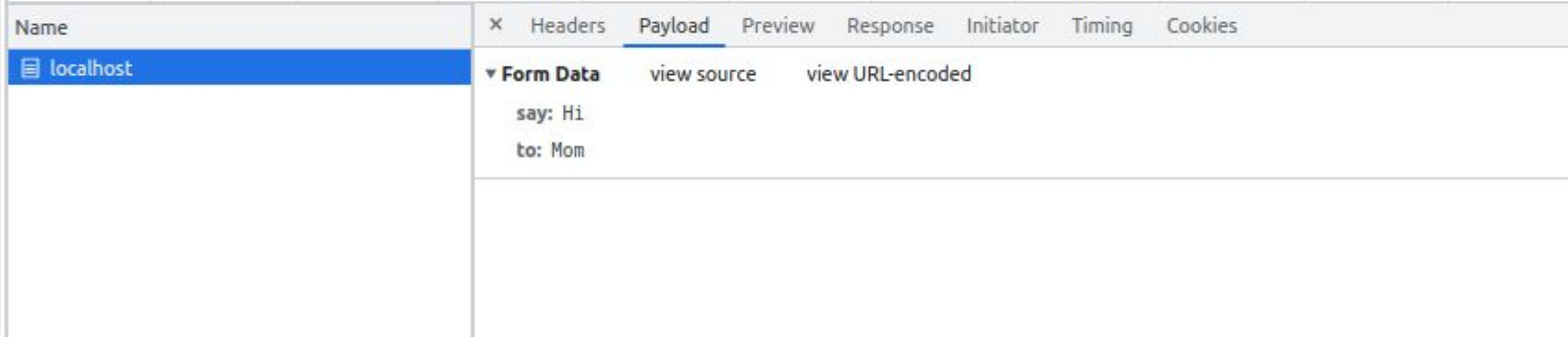
А така изглежда когато се изпрати заявка



POST заявката изпраща и тяло

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <form action="http://localhost:8000/" method="POST">
      <div>
        <label for="say">What greeting do you want to say?</label>
        <input name="say" id="say" value="Hi" />
      </div>
      <div>
        <label for="to">Who do you want to say it to?</label>
        <input name="to" id="to" value="Mom" />
      </div>
      <div>
        <button>Send my greetings</button>
      </div>
    </form>
  </body>
</html>
```





И как създавам REST API?

Expressjs

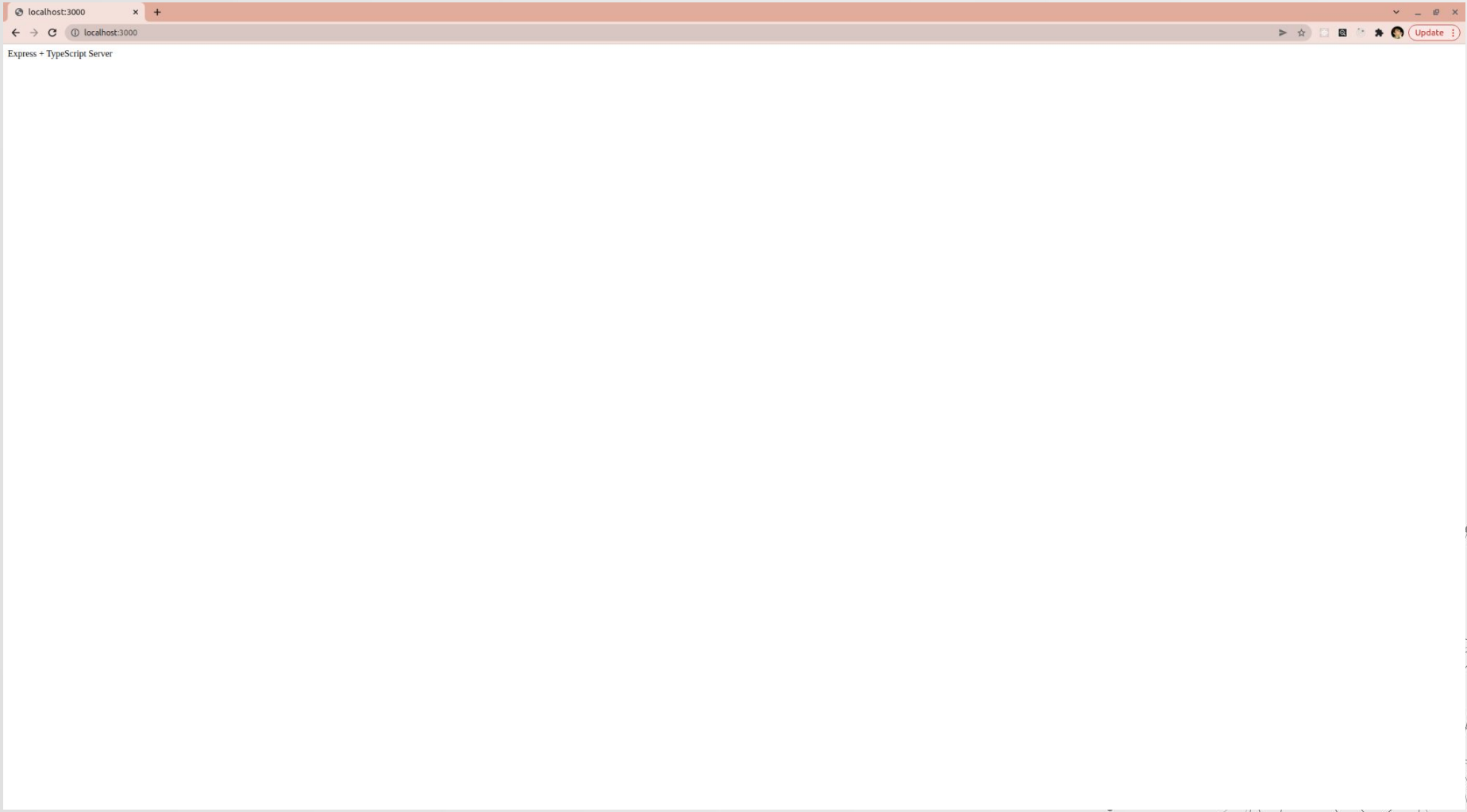
```
iarabadzhiyski@Astea-20158:~$ node -v
v14.18.0
iarabadzhiyski@Astea-20158:~$ npm -v
6.14.15
```

```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ npm init -y
Wrote to /home/iarabadzhiyski/js/lectures/restServices/expressExample/package.json:

{
  "name": "expressexample",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ npm install express

added 50 packages, and audited 51 packages in 2s

2 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

```
1   const express = require("express");
2
3   const app = express();
4   const port = 3000;
5
6   app.get("/", (req, res) => {
7     res.send("Express + TypeScript Server");
8   });
9
10  app.listen(port, () => {
11    console.log(`[server]: Server is running at https://localhost:${port}`);
12  });
13
```

```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ node index.js
[server]: Server is running at https://localhost:3000
```


localhost:3000
localhost:3000
Update
Express + TypeScript Server

```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ npm i -D typescript @types/express @types/node

added 10 packages, and audited 61 packages in 3s

2 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

package.json

```
"devDependencies": {
  "@types/express": "^4.17.13",
  "@types/node": "^17.0.23",
  "typescript": "^4.6.3"
}
```

```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ npx tsc --init

Created a new tsconfig.json with:

  target: es2016
  module: commonjs
  strict: true
  esModuleInterop: true
  skipLibCheck: true
  forceConsistentCasingInFileNames: true


You can learn more at https://aka.ms/tsconfig.json
```

```ts
import express, { Express, Request, Response } from "express";

const app: Express = express();
const port = process.env.PORT;

app.get("/", (req: Request, res: Response) => {
  res.send("Express + TypeScript Server");
});

app.listen(port, () => {
  console.log(`⚡[server]: Server is running at https://localhost:${port}`);
});
```

```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ npm install -D concurrently nodemon

added 131 packages, and audited 192 packages in 10s

20 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

```
  "scripts": {
    "build": "npx tsc",
    "start": "node dist/index.js",
    "dev": "concurrently \"npx tsc --watch\" \"nodemon -q dist/index.js\""
  },
```

Какво друго можем да правим с express?

```
app.get("/", (req: Request, res: Response) => {
  console.log(req.query);
  res.send("Express + TypeScript Server");
});

app.post("/", (req: Request, res: Response) => {
  console.log(req.body);
  res.sendFile(path.join(__dirname, "htmlTemplates/index.html"));
});

app.post("/upload", upload.single("myFile"), (req: Request, res: Response) => {
  if (!req.file) {
    return res.status(401).json({ error: "Please provide an image" });
  }
  const imagePath = path.join(__dirname, `public/${req.file.originalname}`);
  createWriteStream(imagePath).write(req.file.buffer);
  res.redirect(`/media/${req.file.originalname}`);
});
```

What the middleware?

```
app.use(urlencoded({ extended: true }));
app.use(express.json());

function logMethod(req: Request, res: Response, next: NextFunction) {
  console.log("Request Type:", req.method);
  next();
}

app.use((req, res, next) => {
  console.log("Time:", Date.now());
  next();
});

app.use(logMethod);
```

Автентикация?

```
function authenticateJWT(req: Request, res: Response, next: NextFunction) {
  const authHeader = req.headers.authorization;
  if (authHeader && authHeader !== "null") {
    const token = authHeader.split(" ")[1];
    jwt.verify(token, JWT_KEY, (err: any, user: any) => {
      if (err) {
        return res
          .status(403)
          .send({ success: false, message: "Token Expired" });
      }
      // req.user = user;
      next();
    });
  } else {
    res.status(403).json({ success: false, message: "UnAuthorized" });
  }
}
```

Ние искаме RESTful API -> трябва да връщаме JSON.

```
app.get("/", (req: Request, res: Response) => {
  console.log(req.query);
  res.json({ name: "Express + TypeScript Server", ...req.query });
});

app.post("/", (req: Request, res: Response) => {
  console.log(req.body);
  res.json({ ...req.body });
});

app.post("/upload", upload.single("myFile"), (req: Request, res: Response) => {
  if (!req.file) {
    return res.status(401).json({ error: "Please provide an image" });
  }
  const imagePath = path.join(__dirname, `public/${req.file.originalname}`);
  createWriteStream(imagePath).write(req.file.buffer);
  res.json({ path: `http://localhost:8000/media/${req.file.originalname}` });
});
```

Как обикновено изглежда един RESTful API?

```ts
type Book = {
  id?: string;
  createdAt?: number;
  title: string;
  author: string;
  finished: boolean;
};

const harryPotterAndThePhilosophersStone: Book = {
  id: "0",
  title: "Harry Potter and the Philosopher's Stone",
  author: "J. K. Rowling",
  finished: true,
};

const books = [harryPotterAndThePhilosophersStone];

app.get("/books/", (req: Request, res: Response) => {
  res.json({ books });
});

app.get("/books/:id", (req: Request, res: Response) => {
  const book = books.find((book) => book.id === req.params.id);
  if (!book) {
    return res.status(404).send();
  }
  res.json({ book });
});
```

```ts
function createNewBook({
  title,
  author,
  finished,
}: Pick<Book, "title" | "author" | "finished">): Book {
  // automatically create an id, use uuid or let the db generate one
  return {
    id: Math.random().toString(),
    title,
    author,
    finished,
    createdAt: Date.now(),
  };
}

function saveItToTheDatabase(book: Book) {
  return true;
}

app.post("/books", (req: Request, res: Response) => {
  // create a new book and save it to the database
  const book = createNewBook(req.body);
  if (!book) {
    return res.status(400).send();
  }
  const saved = saveItToTheDatabase(book);
  if (!saved) {
    return res.status(400).send();
  }
  res.json({ book });
});
```

Swagger?

Въпроси?